edeposit.amqp.ftp

Release 1.0.2

February 05, 2015

Contents

This module provides wrappers over ProFTPD FTP server for edeposit project.

It allows producers automatic and/or batch uploads of both files and metadata. Metadata are recognized and parsed by this package and in case of error, user is notified by creating special file with error log.

CHAPTER 1

Installation

This module is hosted at PIP, so you can install it easily with following command:

```
sudo pip install edeposit.amqp.ftp
```

This will install the module and all necessary requirements with one exception - the ProFTPD server itself. That can be installed manually or using package manager from your distribution.

Ubuntu/Debian:

```
sudo apt-get install proftpd-basic proftpd-mod-vroot
```

OpenSuse:

```
sudo zypper install proftpd
```

1.1 Initialization

After installation of the `ProFTPD` and `edeposit.amqp.ftp`, run the *edeposit_proftpd_init.py* script (should be in your path), which will configure ProFTPD and create all necessary files and directories.

Depending at which system are you using, you may need to restart/reload the `proftpd` daemon.

You may also want to check `settings` module, to change some of the paths using JSON configuration files.

CHAPTER 2

Usage

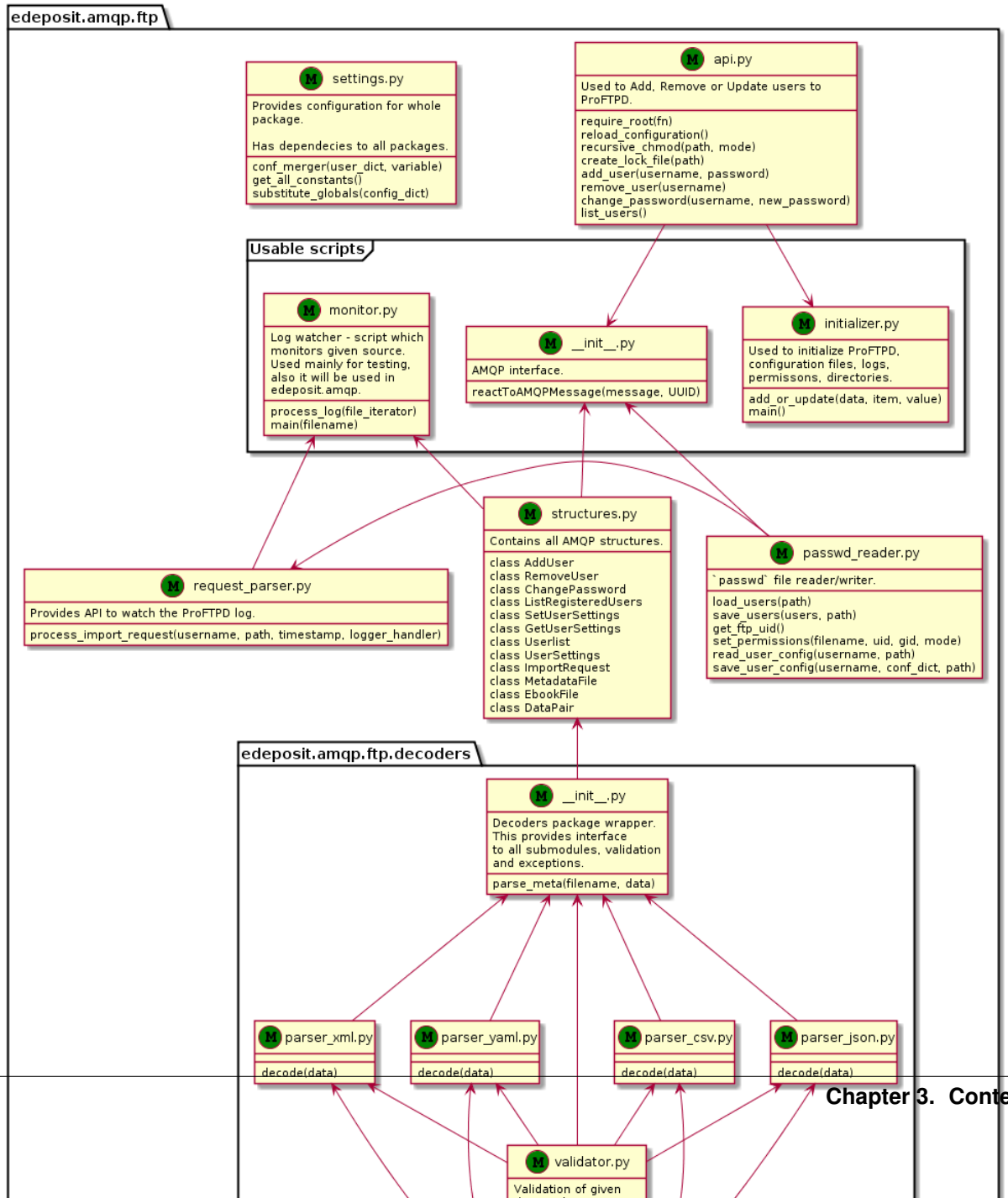
There is guide how to use the package from user perspective:

- `/workflow/workflow` (english version)
- `/workflow/pouziti` (czech version)

CHAPTER 3

Content

Dependencies of submodules



Parts of the module can be divided into two subcategories - scripts and parts of the API.

Scripts are meant to be used by users, API is there mainly for programmers.

3.1 Standalone scripts

3.1.1 Initializer script

This script is used to initialize ProFTPD and set configuration required by edeposit.amqp.ftp module.

It changes/creates ProFTPD configuration file, password file and extened log file. Also user directory is created and correct permissions is set.

Usage

```
$ ./edeposit_proftpd_init.py -h
usage: edeposit_proftpd_init.py [-h] [-o] [-v]

This script will modify your ProFTPD installation for use with
edeposit.amqp.ftp package.

optional arguments:
  -h, --help       show this help message and exit
  -o, --overwrite  Overwrite ProFTPD configuration file with edeposit.amqp.ftp
                   default configuration.
  -v, --verbose    Print debug output.
```

API

edeposit_proftpd_init.**main**(**args*, ***kwargs*)
Used to create configuration files, set permissions and so on.

edeposit_proftpd_init.**add_or_update**(*data*, *item*, *value*)
Add or update value in configuration file format used by proftpd.

Parameters

- **data** (*str*) – Configuration file as string.
- **item** (*str*) – What option will be added/updated.
- **value** (*str*) – Value of option.

Returns updated configuration

Return type str

3.1.2 Monitor script

This script is used to monitor ProFTPD log and to react at certain events (deletion of the `ftp.settings.LOCK_FILENAME`).

It is also used at API level in `edeposit.amqp` (see `process_log()` and `ftp_managerd`).

Details of parsing are handled by `request_parser`.

ftp.monitor.**_read_stdin**()
Generator for reading from standard input in nonblocking mode.

Other ways of reading from `stdin` in python waits, until the buffer is big enough, or until EOF character is sent.

This functions yields immediately after each line.

ftp.monitor.**_parse_line**(*line*)
Convert one line from the extended log to dict.

Parameters **line** (*str*) – Line which will be converted.

Returns dict with `timestamp`, `command`, `username` and `path` keys.

Return type dict

Note: Typical line looks like this:

```
/home/ftp/xex/asd bsd.dat, xex, STOR, 1398351777
```

Filename may contain `,` character, so I am `rsplitting` the line from the end to the beginning.

---

ftp.monitor.**process_log**(*file_iterator*)
Process the extended ProFTPD log.

Parameters **file_iterator** (*file*) – any file-like iterator for reading the log or stdin (see `_read_stdin()`).

Yields *ImportRequest* – with each import.

ftp.monitor.**main**(*filename*)
Open *filename* and start processing it line by line. If *filename* is none, process lines from *stdin*.

3.2 API

3.2.1 __init__.py

This module provides standard interface for AMQP communication as it is defined and used by `edeposit.amqp`.

The interface consists of `reactToAMQPMessage()` function, which receives two parameters - structure and UUID. UUID is not much important, but structure is usually namedtuple containing information what should module do.

After the work is done, `reactToAMQPMessage()` returns a value, which is then automatically transfered back to caller. If the exception is raised, it is also transfered in open and easy to handle way.

edeposit.amqp.ftp

In this module, `reactToAMQPMessage()` is used only for **receiving** commands from the other side. Events caused by FTP users are handled by `monitor.py`.

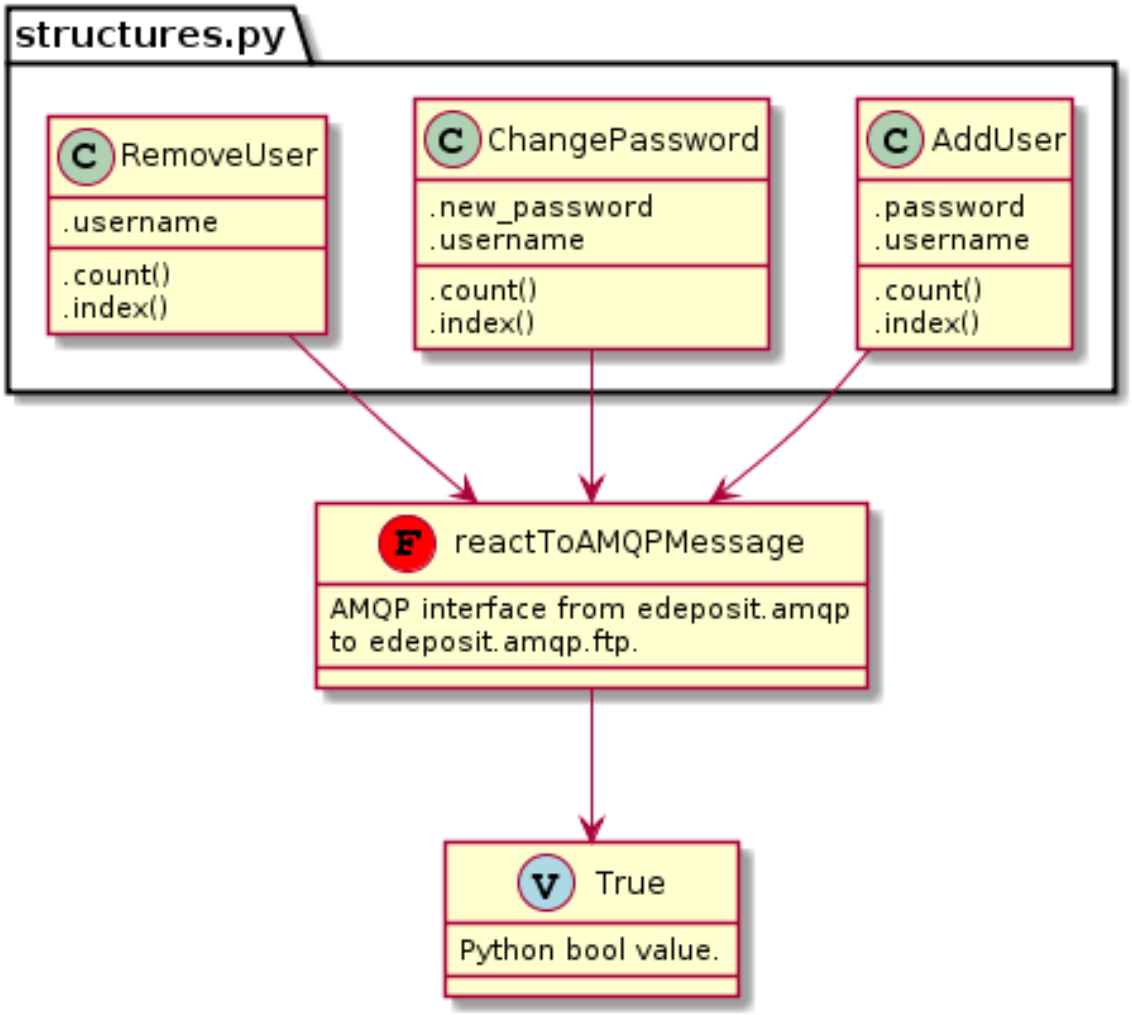
Commands can create/change/remove users and so on. This is done by sending one of the following structures defined in `structures.py`:

- `AddUser`
- `RemoveUser`
- `ChangePassword`

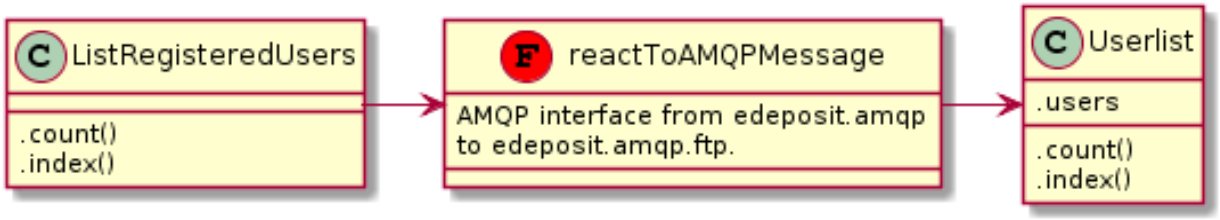
- `ListRegisteredUsers`
- `SetUserSettings`
- `GetUserSettings`

Responses

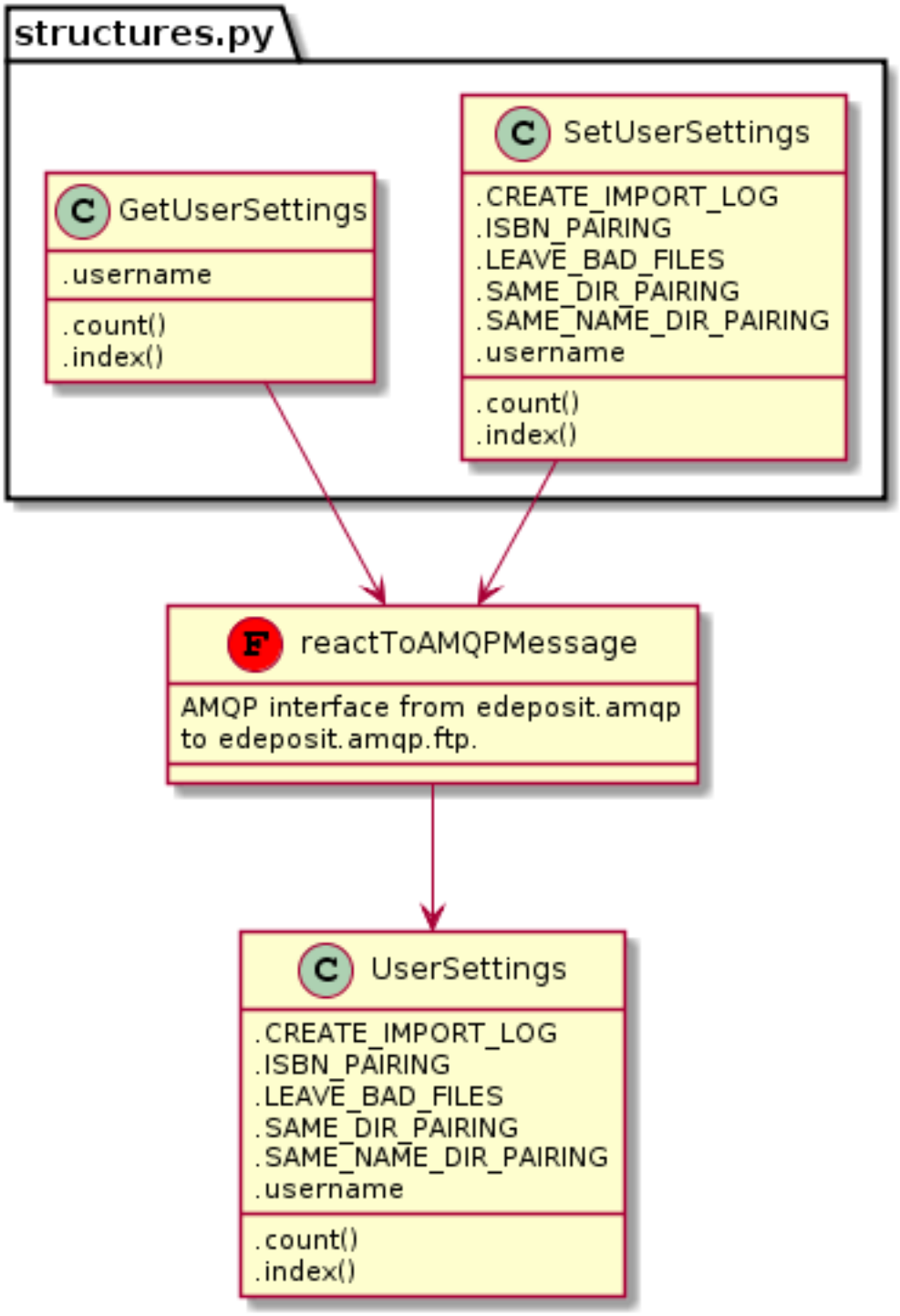
`AddUser`, `RemoveUser` and `ChangePassword` requests at this moment returns just simple `True`. This may be changed later.



`ListRegisteredUsers` returns `Userlist` class.



`SetUserSettings` and `GetUserSettings` both returns `UserSettings` structure.



API

ftp.**reactToAMQPMessage**(*message*, *send_back*)
React to given (AMQP) message. *message* is expected to be `collections.namedtuple()` structure from `structures` filled with all necessary data.

Parameters

- **message** (*object*) – One of the request objects defined in `structures`.
- **send_back** (*fn reference*) – Reference to function for responding. This is useful for progress monitoring for example. Function takes one parameter, which may be response structure/namedtuple, or string or whatever would be normally returned.

Returns Response class from `structures`.

Return type object

Raises `ValueError` – if bad type of *message* structure is given.

3.2.2 Request parser

This submodule provides ability to process and parse import requests.

Most important function in this matter is the `process_import_request()`, which is called from from `ftp.monitor.process_log()`. When it is called, it scans the user's home directory, detects new files, pairs them together into proper objects (see `ftp.structures`, speficifally `MetadataFile`, `EbookFile` and `DataPair`).

API

ftp.request_parser.**process_import_request**(*username*, *path*, *timestamp*, *logger_handler*)
React to import request. Look into user's directory and react to files user uploaded there.

Behavior of this function can be set by setting variables in `ftp.settings`.

Parameters

- **username** (*str*) – Name of the user who triggered the import request.
- **path** (*str*) – Path to the file, which triggered import request.
- **timestamp** (*float*) – Timestamp of the event.
- **logger_handler** (*object*) – Python logger. See `logging` for details.

Returns `:ImportRequest`.

3.2.3 ProFTPD API

ProFTPD wrapped used to manage users of the FTP server.

This module controls the `ftpd.passwd` (`LOGIN_FILE`), creates/removes users directory and so on.

Warning: This API supposes, that it has permissions to read/write to *ProFTPD* configuration directory and to *root* directory for users.

Note: You don't have to set the permissions and everything manually, there is script called `initializer`, which

can do it for you automatically.

---

ftp.api.**require_root**(*fn*)
Decorator to make sure, that user is root.

ftp.api.**reload_configuration**(**args*, ***kwargs*)
Send signal to the proftpd daemon to reload configuration.

ftp.api.**recursive_chmod**(*path*, *mode=493*)
Recursively change `mode` for given `path`. Same as `chmod -R mode`.

Parameters

- **path** (*str*) – Path of the directory/file.
- **mode** (*octal int, default 0755*) – New mode of the file.

Warning: Don't forget to add `0` at the beginning of the numbers of *mode*, or *Unspeakable hOrRoRs* will be awaken from their unholy sleep outside of the reality and they WILL eat your soul (and your files).

ftp.api.**create_lock_file**(*path*)
Create lock file filled with `LOCK_FILE_CONTENT`.

Parameters **path** (*str*) – Path to the lock file. Made from users home directory and `LOCK_FILENAME`.

ftp.api.**add_user**(**args*, ***kwargs*)
Adds record to passwd-like file for ProFTPD, creates home directory and sets permissions for important files.

Parameters

- **username** (*str*) – User's name.
- **password** (*str*) – User's password.

ftp.api.**remove_user**(**args*, ***kwargs*)
Remove user, his home directory and so on..

Parameters **username** (*str*) – User's name.

ftp.api.**change_password**(**args*, ***kwargs*)
Change password for given *username*.

Parameters

- **username** (*str*) – User's name.
- **new_password** (*str*) – User's new password.

ftp.api.**list_users**(**args*, ***kwargs*)
List all registered users, which are stored in `LOGIN_FILE`.

Returns of str usernames.

Return type list

3.2.4 Passwd reader

API for reading/writing of the passwd file used by ProFTPD (and also unix).

API

ftp.passwd_reader.**load_users**(*path='/etc/proftpd/ftpd.passwd'*)
Read passwd file and return dict with users and all their settings.

Parameters **path** (*str, default settings.LOGIN_FILE*) – path of the file, which will be loaded (default `ftp.settings.LOGIN_FILE`).

Returns (dict): username: {pass_hash, uid, gid, full_name, home, shell}

Example of returned data:

```
{
    "xex": {
        "pass_hash": "$asd$aiosjdaiosjdásghwasdjo",
        "uid": "2000",
        "gid": "2000",
        "full_name": "ftftf",
        "home": "/home/ftp/xex",
        "shell": "/bin/false"
    }
}
```

ftp.passwd_reader.**save_users**(*users*, *path='/etc/proftpd/ftpd.passwd'*)
Save dictionary with user data to passwd file (default `ftp.settings.LOGIN_FILE`).

Parameters

- **users** (*dict*) – dictionary with user data. For details look at dict returned from `load_users()`.
- **path** (*str, default settings.LOGIN_FILE*) – path of the file, where the data will be stored (default `ftp.settings.LOGIN_FILE`).

ftp.passwd_reader.**get_ftp_uid**()

Returns UID of the proftpd/ftp user.

Return type int

Raises `KeyError` – When `proftpd` and `ftp` user is not found.

ftp.passwd_reader.**set_permissions**(*filename*, *uid=None*, *gid=None*, *mode=509*)
Set pemissions for given *filename*.

Parameters

- **filename** (*str*) – name of the file/directory
- **uid** (*int, default proftpd*) – user ID - if not set, user ID of *proftpd* is used
- **gid** (*int*) – group ID, if not set, it is not changed
- **mode** (*int, default 0775*) – unix access mode

ftp.passwd_reader.**read_user_config**(*username*, *path='/etc/proftpd/ftpd.passwd'*)
Read user's configuration from otherwise unused field `full_name` in passwd file.

Configuration is stored in string as list of t/f characters.

ftp.passwd_reader.**save_user_config**(*username*, *conf_dict*, *path='/etc/proftpd/ftpd.passwd'*)
Save user's configuration to otherwise unused field `full_name` in passwd file.

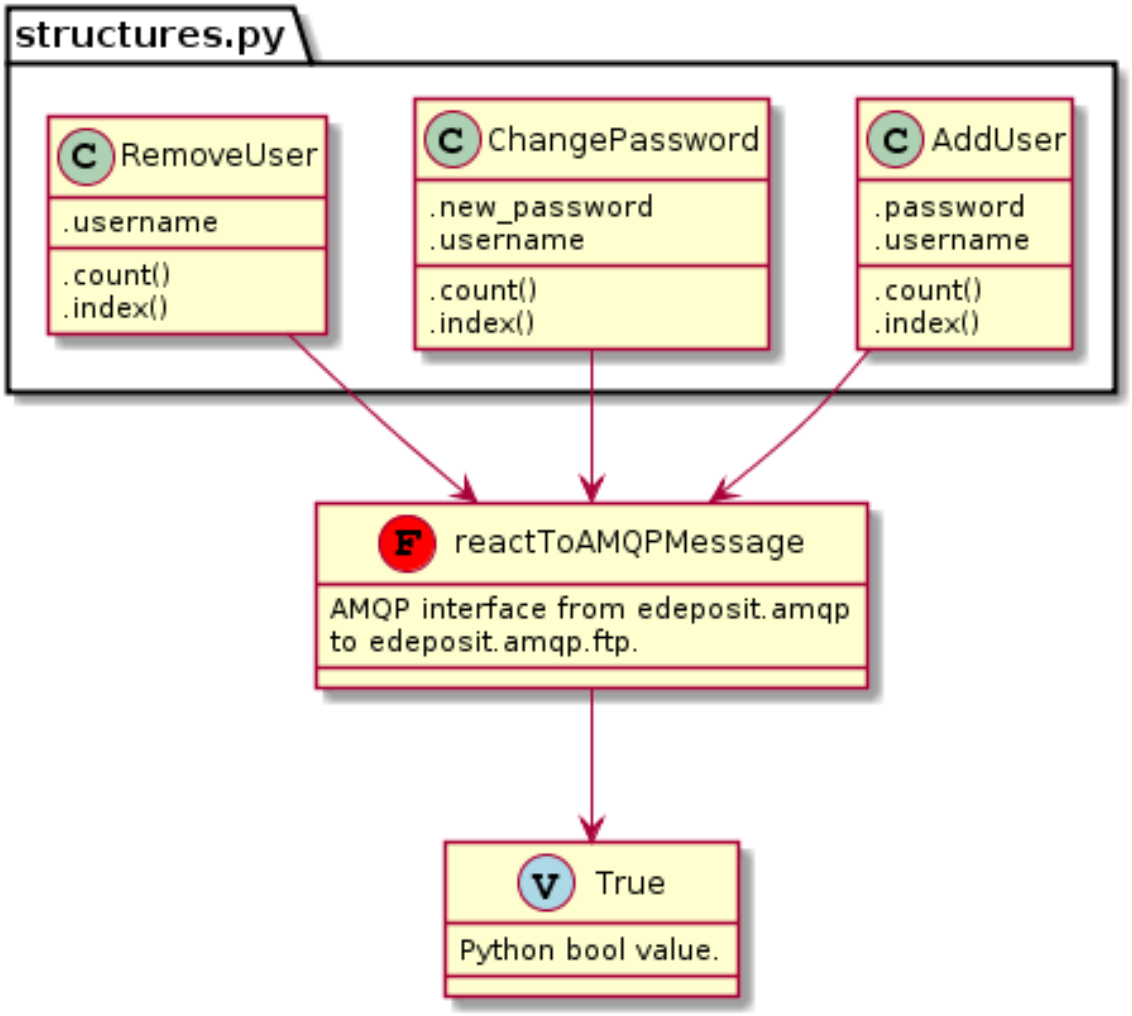
3.2.5 AMQP messages/structures

This module contains all communication structures used in AMQP communication.

Classes from Requests are used to manipulate FTP users.

Requests

User management requests



class `ftp.structures.`**`AddUser`**
Add new user to the ProFTPD server.

Parameters

- **username** (*str*) – Alloed characters: `a-zA-Z0-9._-`.
- **password** (*str*) – Password for the new user. Only hash is stored.

class `ftp.structures.`**`RemoveUser`**
Remove user from the ProFTPD server.

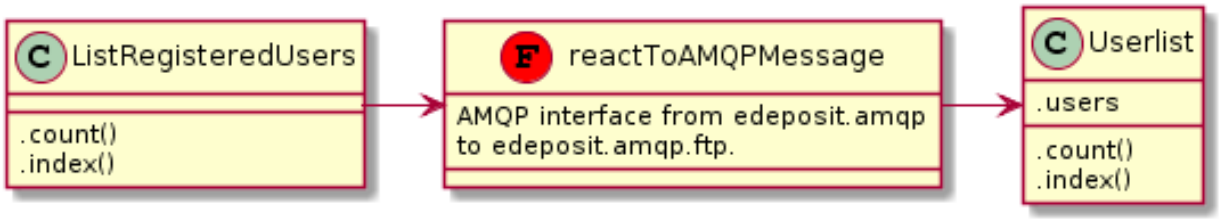
Parameters **username** (*str*) – Alloed characters: `a-zA-Z0-9._-`.

class `ftp.structures.`**`ChangePassword`**
Change password for the user.

Parameters

- **username** (*str*) – Alloed characters: `a-zA-Z0-9._-`.
- **new_password** (*str*) – New password for user.

User requests

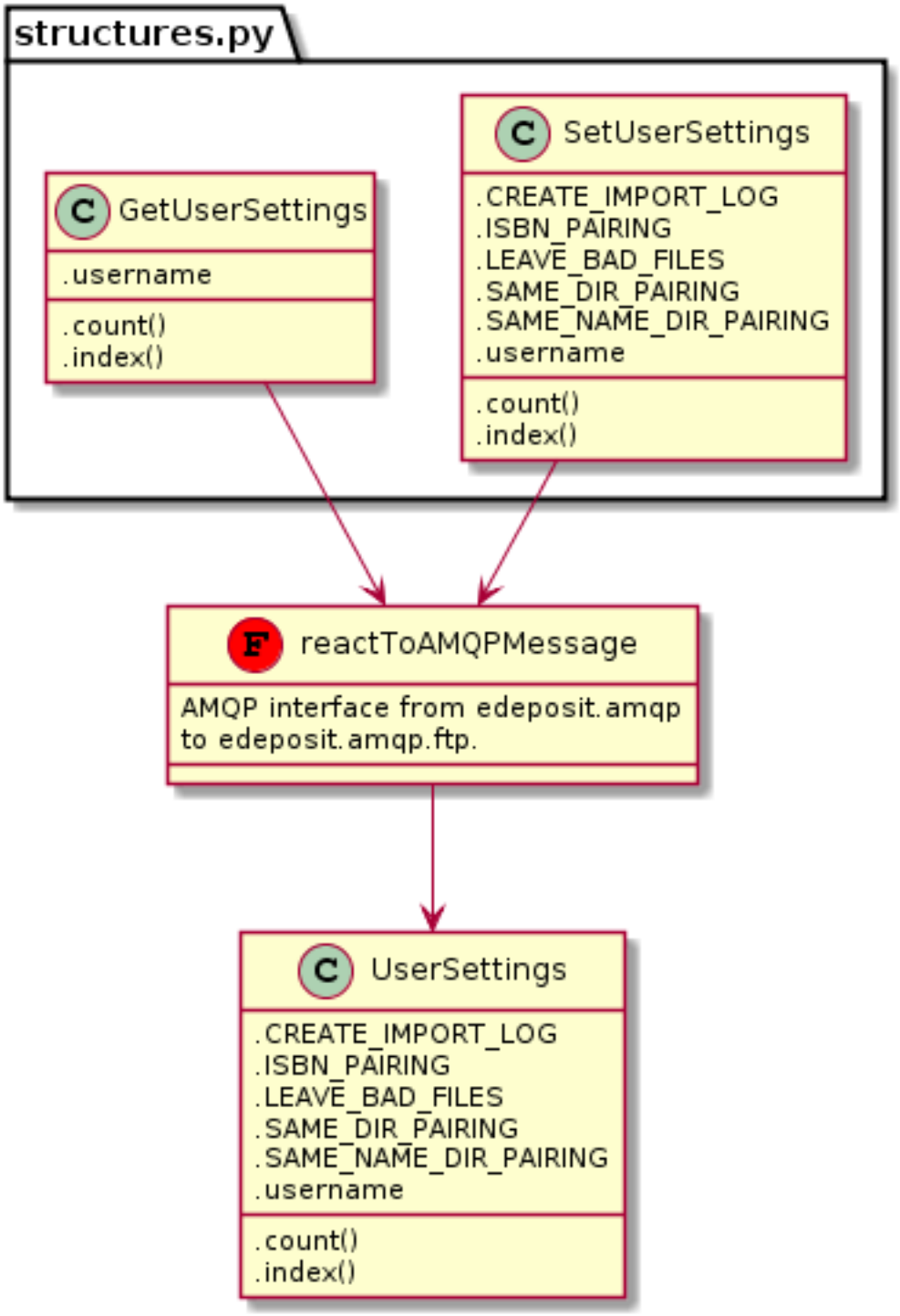


class `ftp.structures.`**`ListRegisteredUsers`**
List all registered users.

See also:

`Userlist`.

Settings management



class `ftp.structures.`**SetUserSettings**

Set settings for the user. `UserSettings` is returned as response.

See also:

UserSettings.

CREATE_IMPORT_LOG
Alias for field number 4

ISBN_PAIRING
Alias for field number 3

LEAVE_BAD_FILES
Alias for field number 5

SAME_DIR_PAIRING
Alias for field number 2

SAME_NAME_DIR_PAIRING
Alias for field number 1

username
Alias for field number 0

class ftp.structures.**GetUserSettings**
Get settings for given *username*.

UserSettings is returned as response.

See also:

UserSettings.

Responses

class ftp.structures.**Userlist**
Response containing names of all users.

users
list

List of registered users.

class ftp.structures.**UserSettings**
All user settings, that user can set himself.

CREATE_IMPORT_LOG
Alias for field number 4

ISBN_PAIRING
Alias for field number 3

LEAVE_BAD_FILES
Alias for field number 5

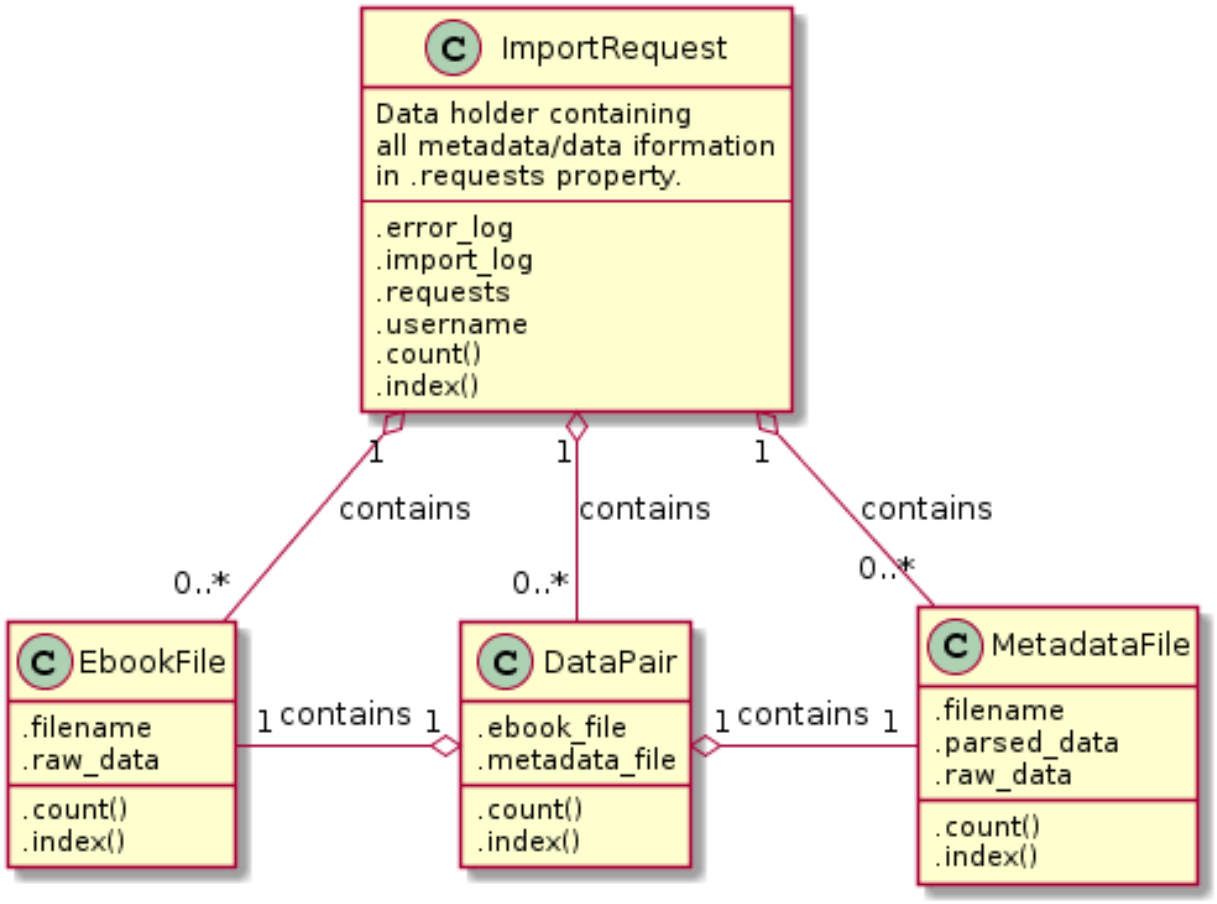
SAME_DIR_PAIRING
Alias for field number 2

SAME_NAME_DIR_PAIRING
Alias for field number 1

username
Alias for field number 0

Import request

Import request are sent by `monitor` itself, without need of programmer interaction.



class `ftp.structures.`**`ImportRequest`**

User's import request - mix of files, metadata and metadata-files pairs.

This request is sent asynchronously when user triggers the upload request.

`username`

str

Name of the user who sent an import request.

`requests`

list

List of `MetadataFile`/`EbookFile`/ `DataPair` objects.

`import_log`

str

Protocol about import.

`error_log`

str

Protocol about errors.

File structures

Following structures may be present in `ImportRequest.requests`.

class `ftp.structures.`**`MetadataFile`**
Structure used to represent Metadata files.

`filename`
str

Name of the parsed file.

`raw_data`
str

Content of the parsed file.

`parsed_data`
EPublication

`EPublication` structure.

class `ftp.structures.`**`EbookFile`**
Structure used to represent data (ebook) files.

`filename`
str

Path to the ebook file.

`raw_data`
str

Content of the file.

class `ftp.structures.`**`DataPair`**
Structure used to repesent MetadataFile - EbookFile pairs.

`metadata_file`
MetadataFile

Metadata.

`ebook_file`
EbookFile

Data.

3.2.6 Decoders submodule

Decoders module used to parser metadata file into `EPublication` structure.

`ftp.decoders.`**`parse_meta`**(*filename*, *data*)
Parse *data* to EPublication.

Parameters

- **filename** (*str*) – Used to choose right parser based at suffix.
- **data** (*str*) – Content of the metadata file.

Returns object.

Return type EPublication

Available decoders

JSON decoder

This submodule is used to parse metadata from JSON (`.json`) files.

Metadata can be stored either in dictionary or in flat array.

Example structure:

```
{
    "ISBN knihy": "80-86056-31-7",
    "Vazba knihy": "brož.",
    "Nazev knihy": "80-86056-31-7.json",
    "Misto vydani": "Praha",
    "Nakladatel": "Garda",
    "Datum vydani": "09/2012",
    "Poradi vydani": "1",
    "Zpracovatel zaznamu": "Franta Putsalek"
}
```

or:

```
[
    "ISBN knihy", "80-86056-31-7",
    "Vazba knihy", "brož.",
    "Nazev knihy", "samename.json",
    "Misto vydani", "Praha",
    "Nakladatel", "Garda",
    "Datum vydani", "09/2012",
    "Poradi vydani", "1",
    "Zpracovatel zaznamu", "Franta Putsalek"
]
```

See `/workflow/required` for list of required fields.

ftp.decoders.parser_json.**decode**(*data*)
Handles decoding of the JSON *data*.

Parameters **data** (*str*) – Data which will be decoded.

Returns Dictionary with decoded data.

Return type dict

XML decoder

This submodule is used to parse metadata from XML (`.xml`) files.

Format schema:

```
<root>
    <item key="key">value</item>
</root>
```

Example of valid data:

```
<root>
    <item key="ISBN knihy">80-86056-31-7</item>
    <item key="Vazba knihy">brož.</item>
```

```
    <item key="Nazev knihy">standalone2.xml</item>
    <item key="Misto vydani">Praha</item>
    <item key="Nakladatel">Garda</item>
    <item key="Datum vydani">09/2012</item>
    <item key="Poradi vydani">1</item>
    <item key="Zpracovatel zaznamu">Franta Putsalek</item>
</root>
```

See `/workflow/required` for list of required fields.

ftp.decoders.parser_xml.**decode**(*data*)
Handles decoding of the XML *data*.

Parameters **data** (*str*) – Data which will be decoded.

Returns Dictionary with decoded data.

Return type dict

CSV decoder

This submodule is used to parse metadata from CSV (`.csv`) files.

Example of the valid data:

```
ISBN knihy;978-80-87270-99-8
Vazba knihy;brož.
Nazev knihy;whatever.csv
Misto vydani;Praha
Nakladatel;Garda
Datum vydani;IX.12
Poradi vydani;1
Zpracovatel zaznamu;Franta Putsalek
```

See `/workflow/required` for list of required fields.

ftp.decoders.parser_csv.**decode**(*data*)
Handles decoding of the CSV *data*.

Parameters **data** (*str*) – Data which will be decoded.

Returns Dictionary with decoded data.

Return type dict

YAML decoder

This submodule is used to parse metadata from YAML (`.yaml`) files.

Example of the valid data:

```
ISBN knihy: 80-86056-31-7
Vazba knihy: brož.
Nazev knihy: 80-86056-31-7.json
Misto vydani: Praha
Nakladatel: Garda
Datum vydani: 09/2012
Poradi vydani: 1
Zpracovatel zaznamu: Franta Putsalek
```

See `/workflow/required` for list of required fields.

`ftp.decoders.parser_yaml.`**`decode`**(*data*)
Handles decoding of the YAML *data*.

Parameters **data** (*str*) – Data which will be decoded.

Returns Dictionary with decoded data.

Return type dict

Other submodules

Validator

This module provides highlevel checking of parsed data for lowlevel decoders.

It handles the unicode in keys, builds dicts from flat arrays and so on.

class `ftp.decoders.validator.`**`Field`**(*keyword*, *descr*, *epub=None*)
This class is used to represent and parse specific "key: val" pair.

When you create the object, *keyword* and *descr* is specified. Optionally also *epub* parameter, which is corresponding key in `EPublication` structure.

Assingning value to the class is done by calling `check()`, which sets the `value`, if the *key* parameter matches `keyword`.

Parameters

- **keyword** (*str*) – Key for the data pair.
- **descr** (*str*) – Description of the data pair. Used in exceptions.
- **epub** (*str, default None*) – Corresponding keyword in EPublication structure.

`keyword` = None
Keyword agains `check()` will try to match.

`descr` = None
Description of the data pair.

`value` = None
Internal value. Set when `check()` successfully matched the keyword.

`epub` = None
Corresponding key in `EPublication` structure.

`check`(*key*, *value*)
Check whether *key* matchs the `keyword`. If so, set the `value` to *value*.

Parameters

- **key** (*str*) – Key which will be matched with `keyword`.
- **value** (*str*) – Value which will be assigned to `value` if keys matches.

Returns True/False: Whether the key matched `keyword`.

`is_valid`()
Return `True` if `value` is set.

Note: `value` is set by calling `check()` with proper *key*.

---

class ftp.decoders.validator.**FieldParser**
Class used to make sure, that all fields in metadata are present.

See /api/required for list of required fields.

process(*key*, *val*)
Try to look for *key* in all required and optional fields. If found, set the *val*.

is_valid()

Returns True/False whether ALL required fields are set.

get_epublication()

Returns Structure when the object is_valid().

Return type EPublication

Raises MetaParsingException – When the object is not valid.

ftp.decoders.validator.**check_structure**(*data*)
Check whether the structure is flat dictionary. If not, try to convert it to dictionary.

Parameters **data** – Whatever data you have (dict/tuple/list).

Returns When the conversion was successful or *data* was already *good*.

Return type dict

Raises MetaParsingException – When the data couldn't be converted or had *bad* structure.

Decoders exceptions

Exceptions for decoders submodule.

exception ftp.decoders.meta_exceptions.**MetaParsingException**(*message*)
Bases: exceptions.UserWarning

Main exception used in every decoder.

Note: You souldn't get anything else from the whole decoders submodule.

3.2.7 Settings and configuration

Module is containing all necessary global variables for the package.

Module also has the ability to read user-defined data from two paths:

- $HOME/_SETTINGS_PATH
- /etc/_SETTINGS_PATH

See _SETTINGS_PATH for details.

Note: If the first path is found, other is ignored.

Example of the configuration file ($HOME/edeposit/ftp.json):

```
{
    "CONF_PATH": "/home/bystrousak/.ftpdconf/"
}
```

Attributes

ftp.settings.**BASE_PATH = '/var/build/user_builds/edeposit-amqp-ftp/checkouts/stable/src/edeposit/amqp/ftp'**
Module's path.

ftp.settings.**CONF_PATH = '/etc/proftpd/'**
Proftpd configuration directory.

ftp.settings.**LOG_PATH = '/var/log/proftpd/'**
Proftpd log directory.

ftp.settings.**DATA_PATH = '/home/ftp/'**
Path to directory, where the user directories will be created.

ftp.settings.**SERVER_ADDRESS = 'localhost'**
Server's address - used only in unit/integration testing.

ftp.settings.**CONF_FILE = '/etc/proftpd/proftpd.conf'**
Proftpd configuration file (in CONF_PATH directory).

ftp.settings.**LOGIN_FILE = '/etc/proftpd/ftpd.passwd'**
File where the login informations will be stored (CONF_PATH is used as dirname).

ftp.settings.**LOG_FILE = '/var/log/proftpd/extended.log'**
File where the extended logs are stored (LOG_PATH is used as dirname).

ftp.settings.**LOCK_FILENAME = 'delete_me_to_import_files.txt'**
Filename for the locking mechanism.

ftp.settings.**USER_ERROR_LOG = 'error.log.txt'**
Filename, where the error protocol is stored.

ftp.settings.**USER_IMPORT_LOG = 'import.log.txt'**
Filename, where the import protocol for the user is stored.

ftp.settings.**LOCK_FILE_CONTENT = "Delete this file to start batch import of all files, you've uploaded to the server.\n\nS**
Text, which will be writen to the PROTFPD_LOCK_FILENAME.

ftp.settings.**SAME_NAME_DIR_PAIRING = True**
True - will pair files with same filename in same directory

ftp.settings.**SAME_DIR_PAIRING = True**
True - will pair files with different filenames, if there is only two files in dir

ftp.settings.**ISBN_PAIRING = True**
True - if the name is ISBN, files will be paired no matter where they are stored (unless they weren't paired before)

ftp.settings.**LOCK_ONLY_IN_HOME = True**
True - Lock file can be only in home directory, everywhere else will be ignored

ftp.settings.**CREATE_IMPORT_LOG = True**
True - USER_IMPORT_LOG will be created

ftp.settings.**LEAVE_BAD_FILES = True**
True - don't remove badly formatted metadata files

ftp.settings.**PROFTPD_USERS_GID = 2000**
I am using GID 2000 for all our users - this GID shouldn't be used by other than FTP users!

ftp.settings.**conf_merger**(*user_dict*, *variable*)
Merge global configuration with user's personal configuration.

Global configuration has always higher priority.

ftp.settings.**get_all_constants**()
Get list of all uppercase, non-private globals (doesn't start with _).

Returns Uppercase names defined in *globals()* (variables from this module).

Return type list

ftp.settings.**substitute_globals**(*config_dict*)
Set global variables to values defined in *config_dict*.

Parameters **config_dict** (*dict*) – dictionary with data, which are used to set *globals*.

Note: *config_dict* have to be dictionary, or it is ignored. Also all variables, that are not already in globals, or are not types defined in _ALLOWED (str, int, float) or starts with _ are silently ignored.

CHAPTER 4

Source code

The project is opensource (GPL) and source codes can be found at GitHub:

- https://github.com/edeposit/edeposit.amqp.ftp

CHAPTER 5

Testing

Almost every feature of the project is tested in unit/integration tests. You can run this tests using provided `run_tests.sh` script, which can be found in the root of the project.

5.1 Requirements

This script expects that pytest is installed. In case you don't have it yet, it can be easily installed using following command:

```
pip install --user pytest
```

or for all users:

```
sudo pip install pytest
```

5.2 Options

Script provides three options - to run just unittests (`-u` switch), to run integration tests (`-i` switch) or to run both (`-a` switch).

Integration tests requires that ProFTPD is installed (there is test to test this) and also **root permissions**. Integration tests are trying all usual (and some unusual) use-cases, permissions to read/write into ProFTPD configuration files and so on. Thats why the root access is required.

Example of the success output from test script:

```
$ ./run_tests.sh -a
[sudo] password for bystrousak:
============================= test session starts ==============================
platform linux2 -- Python 2.7.5 -- py-1.4.20 -- pytest-2.5.2
collected 42 items

src/edeposit/amqp/ftp/tests/integration/test_api.py .....
src/edeposit/amqp/ftp/tests/integration/test_monitor.py .......
src/edeposit/amqp/ftp/tests/unittests/test_settings.py .....
src/edeposit/amqp/ftp/tests/unittests/test_structures.py ...
src/edeposit/amqp/ftp/tests/unittests/test_unit_monitor.py .
src/edeposit/amqp/ftp/tests/unittests/test_unit_passwd_reader.py .....
src/edeposit/amqp/ftp/tests/unittests/test_unit_request_parser.py .....
src/edeposit/amqp/ftp/tests/unittests/test_decoders/test_init.py .
```

```
src/edeposit/amqp/ftp/tests/unittests/test_decoders/test_meta_exceptions.py .
src/edeposit/amqp/ftp/tests/unittests/test_decoders/test_validator.py .....
src/edeposit/amqp/ftp/tests/unittests/test_decoders/test_parser_csv.py .
src/edeposit/amqp/ftp/tests/unittests/test_decoders/test_parser_json.py .
src/edeposit/amqp/ftp/tests/unittests/test_decoders/test_parser_xml.py .
src/edeposit/amqp/ftp/tests/unittests/test_decoders/test_parser_yaml.py .

========================== 42 passed in 13.96 seconds ==========================
```

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*

e

f

Symbols

A

B

C

D

E

F

G

I